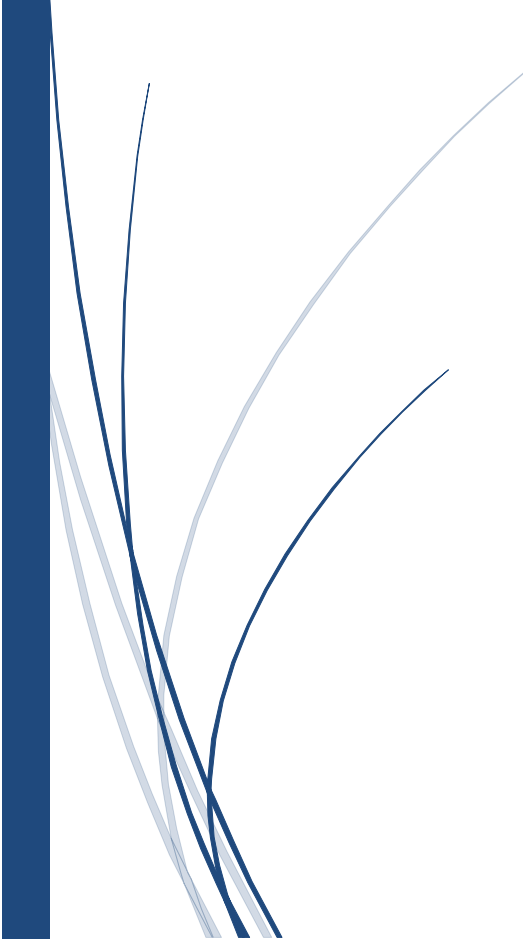




10/16/2019

# ISM 6218: Advanced Database

EXAM - 2



Ezema, Chukwuebuka  
TEAM ENJOY

# Table of Contents

S. No	Description	Page No
1	Introduction	2
2	Question 1 – User Story 1 using CTE	3
3	Question 1 – User Story 2 using SPROC	4
4	Question 1 – User Story 3 using Cursor	5
5	Question 2 – Parameterized SQL query	7
6	Question 3 – Correlated and Uncorrelated sub query	11
7	Question 4 – Case Expressions	12

## 1. Create 3 User Defined Functions, using all the following:

- Union, Union ALL, Coalesce, is null, exist, having, IN, Any
- Except and Intersect operators
- 1 must return a table, 2 must return a single value
- Call 1 UDF from a SPROC
- Call 1 UDF from Cursor
- Call 1 UDF from CTE
- Use Apply operator to join UDF with another table

### User Story 1:

As the BAIS Academic Advisor, I want a report of all the non traditional students with their current GPA, so that I can evaluate it against the class average.

Requirements for user story:

- Create a view of all the non-traditional students Student who are older than 24.
- Get course average
- Check the performance using a scalar UDF.
- Fetch all the student performance info using a CTE.

Used: CTE along with scalar UDF and Operators (IS NULL, Having, IN, ANY, EXCEPT)

The image displays three screenshots of SQL Server Enterprise Manager, showing different SQL queries and their results.

**SQLQuery7.sql - D:\MRA\coezema (51)\***

```
1 use StudentEnrollment;
2
3 With Non_TradPerf_CTE (StudentNo, Name, Age, Major, Performance)
4 AS (
5     Select
6         StudentNo,
7         Name,
8         Age,
9         Major,
10        dbo.getNonTradCourse(StudentNo, Major) as Performance
11    from Student.StudentInfo
12    where Age > 24
13 )
14
15
16 select * from Non_TradPerf_CTE;
17
```

**Results:**

StudentNo	Name	Age	Major	Performance
2	Emmanuel	30	HR	May need attent
3	Nathaniel	36	IT	May need attent
4	Ivan	40	BAIS	May need attent
6	Amos	28	BAIS	May need attent
7	Tucker	25	BAIS	May need attent
8	Raphael	30	HR	May need attent
10	Alfonso	25	CS	Doing Well
11	Eric	31	HR	May need attent
12	Elliott	28	BAIS	Doing Well
13	Harlan	34	HR	May need attent
16	Craig	25	CS	May need attent
17	Demetrius	26	Finance	May need attent
18	Vernon	26	CS	May need attent

**SQLQuery59.sql - D:\MRA\coezema (52)\***

```
1 create view nonTrad
2 as
3 select s.StudentNo
4 from Student.StudentInfo s
5 where s.Age > 24
6
7 except
8 select c.studentNo
9 from Course.CourseGrade c
10 group by StudentNo
11 having avg(Percentage) < 80
```

**Messages:**

Commands completed successfully.

Completion time: 2019-10-16T10:14:02.8577557-04:00

**SQLQuery8.sql - D:\MRA\coezema (61)\***

```
1 create function dbo.getNonTradCourse
2 (@StudentID smallint,@Major varchar(15))
3 Returns varchar (15)
4 Begin
5     Declare @return varchar(15);
6     If (@StudentID = ANY (select s.studentNo
7         from Student.StudentInfo s
8         where s.Age > 24))
9     SET @return = 'Doing Well';
10     If (@return IS NULL)
11     SET @return = 'May need attention';
12     Return @return;
13 END
```

**Messages:**

Commands completed successfully.

Completion time: 2019-10-16T10:17:35.6555256-04:00

**Results:**

From the CTE results, we can see that:

- There are quite a few students that may need our attention, this may be due to some of our students just starting in the program.

## User Story 2:

As the academic advisor I want a report of the non traditional students from Human resource and Finance department who failed their exam so that I can contact them about possible academic probation.

Requirements for user story:

- Get HR and Finance non traditional student data
- Get contact details(email/phone) for each such student.

Used: SPROC along with table-valued UDF and operators (Coalesce, UNION, UNION ALL) and Cross Apply

```
SQLQuery10.sql - ...RMRA\coezema (54)* X
1 CREATE FUNCTION getPoorNTradStudents (@StudentNo varchar (50))
2 Returns TABLE
3 AS RETURN
4 SELECT * FROM Course.CourseGrade AS GRADE
5 WHERE GRADE.StudentNo = @StudentNo
6 AND GRADE.Grade = 'F';
7
8 CREATE PROC GetStudentsContact
9 AS SELECT S.StudentNo, S.Name, COALESCE (S.Email, S.PhoneNo) as 'Contact Info', s.Major, grade
10 FROM Student.StudentInfo AS S Cross Apply getPoorNTradStudents (S.StudentNo) as Grade
11 where s.Major = 'HR'
12 and s.Age>24
13 UNION
14 SELECT S.StudentNo, S.Name, COALESCE (S.Email, S.PhoneNo) as 'Contact Info', s.Major, grade
15 FROM Student.StudentInfo AS S Cross Apply getPoorNTradStudents (S.StudentNo) as Grade
16 where s.Major = 'Finance'
17 and s.Age>24
18 Order BY S.StudentNo
19 go
20 exec GetStudentsContact

SQLQuery63.sql - ...RMRA\coezema (51)* X
1
2 CREATE PROC GetStudentsContactALL
3 AS SELECT S.StudentNo, S.Name, COALESCE (S.Email, S.PhoneNo) as 'Cont
4 FROM Student.StudentInfo AS S Cross Apply getPoorNTradStudents (S.
5 where s.Major = 'HR'
6 and s.Age>24
7 UNION ALL
8 SELECT S.StudentNo, S.Name, COALESCE (S.Email, S.PhoneNo) as 'Cont
9 FROM Student.StudentInfo AS S Cross Apply getPoorNTradStudents (S.
10 where s.Major = 'Finance'
11 and s.Age>24
12 Order BY S.StudentNo
13 go
14 exec GetStudentsContactALL

Results
StudentNo Name Contact Info Major grade
1 8 Raphael 890163507 HR F
2 11 Eric uma.Nunc@facilisSuspensisecommodo.co.uk HR F
3 17 Demetrius mauris.a.nunc@famesacturpis.net Finance F
4 24 Magee 833922965 Finance F
5 39 Jemaine faucibus.leo@eros.co.uk Finance F
6 48 William ullamcorper@nisiMauris.co.uk HR F
7 64 Tate Proin@posuereenim.org HR F
8 84 Zephania lacus@faucibuslectusa.co.uk HR F
9 90 Seth tristique.neque@Nuncullamcorper.ca HR F
10 101 Lila consequat.enim@vulputatelacus.org HR F
11 105 Alea et.magnis@mollisPhaselluslibero.co.uk Finance F
12 109 Ilana Integer.vitae@congue.net Finance F
13 125 Zelda semper.pretium.neque@acmattisvelit.net Finance F

Query execu... DESKTOP-CD9RMRA (14.0 RTM) DESKTOP-CD9RMRA\coezem... StudentEnrollment 00:00:00 15 rows
```

StudentNo	Name	Contact Info	Major	grade
8	Raphael	890163507	HR	F
11	Eric	uma.Nunc@facilisSuspensisecommodo.co.uk	HR	F
17	Demetrius	mauris.a.nunc@famesacturpis.net	Finance	F
24	Magee	833922965	Finance	F
39	Jemaine	faucibus.leo@eros.co.uk	Finance	F
48	William	ullamcorper@nisiMauris.co.uk	HR	F
64	Tate	Proin@posuereenim.org	HR	F
84	Zephania	lacus@faucibuslectusa.co.uk	HR	F
90	Seth	tristique.neque@Nuncullamcorper.ca	HR	F
101	Lila	consequat.enim@vulputatelacus.org	HR	F
105	Alea	et.magnis@mollisPhaselluslibero.co.uk	Finance	F
109	Ilana	Integer.vitae@congue.net	Finance	F
125	Zelda	semper.pretium.neque@acmattisvelit.net	Finance	F

## Results:

- Total 15 Non tradition students received an F and are in jeopardy of getting an academic probation.

What happens if we use UNION ALL instead of UNION?

## Results:

- It returned 17 records.
- Union ALL included duplicate records as well.

## User Story 3:

The dean of the university wants a report of every instructor's class performance to evaluate difficulty level and how students are performing in them.

Requirements of the story:

- Create a function to classify the instructors based on their course difficulty- average grade for student
- Call the function from a Cursor.

**Used: Cursor along with scalar UDF and operators (INTERSECT, EXISTS, IN, IS NULL)**

Let's start by creating a scalar UDF which will classify the class performance for every instructor. If the average class performance is above 70%, then course difficulty is deemed appropriate.

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the Object Explorer shows the database structure, with 'dbo.GetClass' highlighted under 'Scalar-valued Functions'. The main window shows the SQL Query window with the following T-SQL code:

```
1 CREATE FUNCTION dbo.GetClass (@InstructorNo smallint)
2 returns varchar(50)
3 AS
4 BEGIN
5     DECLARE @ret VARCHAR (50);
6     DECLARE @instructorTable TABLE (instructorNo smallint);
7
8     INSERT INTO @instructorTable
9     SELECT Instructor.InstructorInfo.InstructorNo AS 'InstructorID'
10    FROM Instructor.InstructorInfo
11
12    INTERSECT
13    SELECT Course.InstructorNo 'InstructorID'
14    FROM Course.CourseInfo as course
15    where EXISTS (SELECT *
16                  FROM Course.COURSEGRADE as Grade
17                  Group BY Grade. CourseNo
18                  Having course.CourseNo = Grade.CourseNo
19                        AND Avg(Grade.Percentage) > 70)
20
21    IF (@InstructorNo IN (SELECT * FROM @instructorTable))
```

The Messages window at the bottom shows an error:

```
Msg 2714, Level 16, State 3, Procedure GetClass, Line 1 [Batch Start Line 0]
There is already an object named 'GetClass' in the database.

Completion time: 2019-10-16T10:51:52.5488787-04:00
```

Let's now call the function from a cursor.

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the Object Explorer shows the database structure, including tables, views, and functions. The main window shows a T-SQL query in the SQL Query window, which is titled 'SQLQuery17.sql - ...RMRA\coezema (52))'. The query is as follows:

```

1 DECLARE @RESULT AS TABLE(
2     InstructorNo smallint,
3     ClassPerformance varchar(50)
4     Primary Key(InstructorNo));
5
6 DECLARE @InstructorNo smallint,
7         @classPerformance varchar(50);
8
9 DECLARE cursor_faculty CURSOR FOR
10 SELECT I.InstructorNo, dbo.GetClass(I.InstructorNo) from Instructor.InstructorInfo as I;
11
12 OPEN cursor_faculty;
13 FETCH next FROM cursor_faculty INTO @InstructorNo, @classPerformance;
14 WHILE @@FETCH_STATUS = 0
15 BEGIN
16     INSERT INTO @RESULT VALUES (@InstructorNo, @classPerformance);
17     FETCH next FROM cursor_faculty INTO @InstructorNo, @classPerformance;
18 END;
19 CLOSE cursor_faculty;
20 DEALLOCATE cursor_faculty;

```

Below the query, the Results tab shows the output of the query. The results are displayed in a table with two columns: 'InstructorNo' and 'ClassPerformance'. The table contains 14 rows of data, with the first two rows highlighted in yellow.

InstructorNo	ClassPerformance
1001	Students are performing well in this course
1002	Satisfactory but may need additional resources
1003	Satisfactory but may need additional resources
1004	Students are performing well in this course
1005	Students are performing well in this course
1006	Satisfactory but may need additional resources
1007	Satisfactory but may need additional resources
1009	Satisfactory but may need additional resources
1010	Satisfactory but may need additional resources
1011	Satisfactory but may need additional resources
1012	Satisfactory but may need additional resources
1013	Satisfactory but may need additional resources
1014	Satisfactory but may need additional resources

At the bottom of the window, a status bar indicates 'Query executed successfully.' and the server name 'DESKTOP-CD9RMRA (14.0 RTM)'.

## Results:

- Every instructor's class performance has been classified as needing additional resources or students are performing well. If the course average is below 70% then we are not getting the nice bell curve that denotes a normal level of difficulty. There seems to be quite a few courses that may need additional resource and further evaluation will be required prior to committing t

## 2. Create a Parameterized SQL Query

- Use Exec SQL
- Use @SQL

# User Case Story

As the Program Director, I want to know if each course's difficulty is appropriate for students, so that additional resources can be provided if needed.

### STORY REQUIREMENTS:

- Course difficulty is determined by average grade of course
- Course is considered difficult/challenging if mean is below 70
- Course Name should be displayed along with course average
- Difficulty status should be printed

### Using Exec SQL

The screenshot shows a SQL Server Enterprise Manager window with a T-SQL query named 'SQLQuery52.sql'. The query is designed to determine the difficulty status of a course based on its average grade. It uses variables for the course number, average grade, and status. The query logic is as follows:

```
1 DECLARE @sql varchar(max), @columnList varchar(75), @classAvg numeric(5,2)
2 DECLARE @coursePop int
3 DECLARE @course int
4 DECLARE @status varchar(255)
5
6 SET @course = 10004
7 SET @classAvg = (SELECT avg(percentage) from Course.CourseGrade WHERE CourseNo = @course)
8 SET @coursePop = (SELECT count(StudentNo) from Course.CourseGrade WHERE CourseNo = @course)
9 SET @columnList = 'c.CourseNo,ci.Name,'
10
11
12 IF ((@classAvg) < 70.00) SET @status = ''Students find this course challenging''
13 ELSE SET @status = ''Difficulty Level is appropriate''
14
15 SET @sql = 'SELECT ' + @columnList + cast(@classAvg as varchar(255))+ 'as Class'+',' + @status+ ' Course_Status'+
16 ' FROM Course.CourseGrade c join Course.CourseInfo ci
17 on c.CourseNo = ci.CourseNo
18 where c.CourseNo = ' + cast(@course as varchar(255))+
19 'group by c.CourseNo, ci.Name'
20 EXEC(@sql)
```

The results pane shows a single row of data for course 10004:

CourseNo	Name	Class	Course_Status	
1	10004	Analytical Methods for Business	63.82	Students find this course challenging

Logic check using course 10005.

The screenshot shows the results of a logic check for course 10005. The results pane displays a single row of data:

CourseNo	Class Average	Course_Status	
1	10005	74.31	Student are able to perform optimally



## Using @SQL:

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows a T-SQL query in a script editor. The query declares variables for a course number, average grade, student count, and status. It then sets these variables based on data from the 'Course.CourseGrade' table for course 10004. An IF statement checks if the average grade is below 70.00 to set the status. Finally, it constructs and executes a dynamic SQL query that joins 'Course.CourseGrade' and 'Course.CourseInfo' to produce the results shown in the bottom pane.

```
1 DECLARE @sql nvarchar(max), @columnList varchar(75), @classAvg numeric(5,2)
2 DECLARE @coursePop int
3 DECLARE @course int
4 DECLARE @status varchar(255)
5
6 SET @course = 10004
7 SET @classAvg = (SELECT avg(percentage) from Course.CourseGrade WHERE CourseNo = @course)
8 SET @coursePop = (SELECT count(StudentNo) from Course.CourseGrade WHERE CourseNo = @course)
9 SET @columnList = 'c.CourseNo,ci.Name,'
10
11
12 IF ((@classAvg) < 70.00) SET @status = ''Students find this course challenging''
13 ELSE SET @status = ''Difficulty Level is appropriate''
14
15 SET @sql = 'SELECT ' + @columnList + cast(@classAvg as varchar(255))+'as Class'+',' + @status+' Course_Status'+
16           ' FROM Course.CourseGrade c join Course.CourseInfo ci
17           on c.CourseNo = ci.CourseNo
18           where c.CourseNo = @course
19           group by c.CourseNo, ci.Name'
20
21 EXECUTE sp_executesql @sql, N'@course varchar(75)', @course = @course
```

91 %

Results Messages

	CourseNo	Name	Class	Course_Status
1	10004	Analytical Methods for Business	63.82	Students find this course challenging

## Results:

- The Program director can quickly assess the class performance by inserting the course No. if students are performing poorly on average, then the director can make additional resources like TA 's and GA available for students to boost the class average.

### 3. Create 2 Sub queries: 1 correlated, 1 uncorrelated

**User story:** Program director wants a report of all the nontraditional students (age at least 10 years greater than the average in the major/dept) for the USNEWS college population reporting.

- **Correlated Subquery**

**Uncorrelated Subquery**

The screenshot displays two SQL queries in SQL Server Enterprise Manager, comparing a correlated subquery with an uncorrelated subquery. The Object Explorer on the left shows the database structure, including tables like StudentInfo and StudentEnrollment.

**Correlated Subquery (SQLQuery15.sql):**

```
1 /*Correlated subquery*/
2
3
4 select StudentNo, Name, Major, Age
5 from Student.StudentInfo Greater
6 where Age >
7     (Select (AVG(cast(Age as int) + 10))
8      from Student.StudentInfo average
9      where Greater.Major = average.Major)
10
```

**Uncorrelated Subquery (SQLQuery14.sql):**

```
1 /*Uncorrelated sub query*/
2
3 select StudentNo, Name, Major, Age
4 from Student.StudentInfo
5 where Age >
6     (
7      Select (AVG(cast(Age as int)) + 10) from Student.StudentInfo
8     )
9 Order By Major;
```

**Results:**

**Correlated Subquery Results (7 rows):**

StudentNo	Name	Major	Age
4	Ivan	BAIS	40
44	Simon	BAIS	40
175	Jenna	BAIS	40
167	Mariam	CS	39
168	Mercedes	CS	39
56	Uriel	Finance	40
85	Jackson	HR	39

**Uncorrelated Subquery Results (12 rows):**

StudentNo	Name	Major	Age
4	Ivan	BAIS	40
44	Simon	BAIS	40
175	Jenna	BAIS	40
177	Christine	BAIS	39
167	Mariam	CS	39
168	Mercedes	CS	39
56	Uriel	Finance	40
158	Juliet	Finance	39
85	Jackson	HR	39
152	Patricia	IT	39
59	Malcolm	IT	39
34	Herman	IT	39

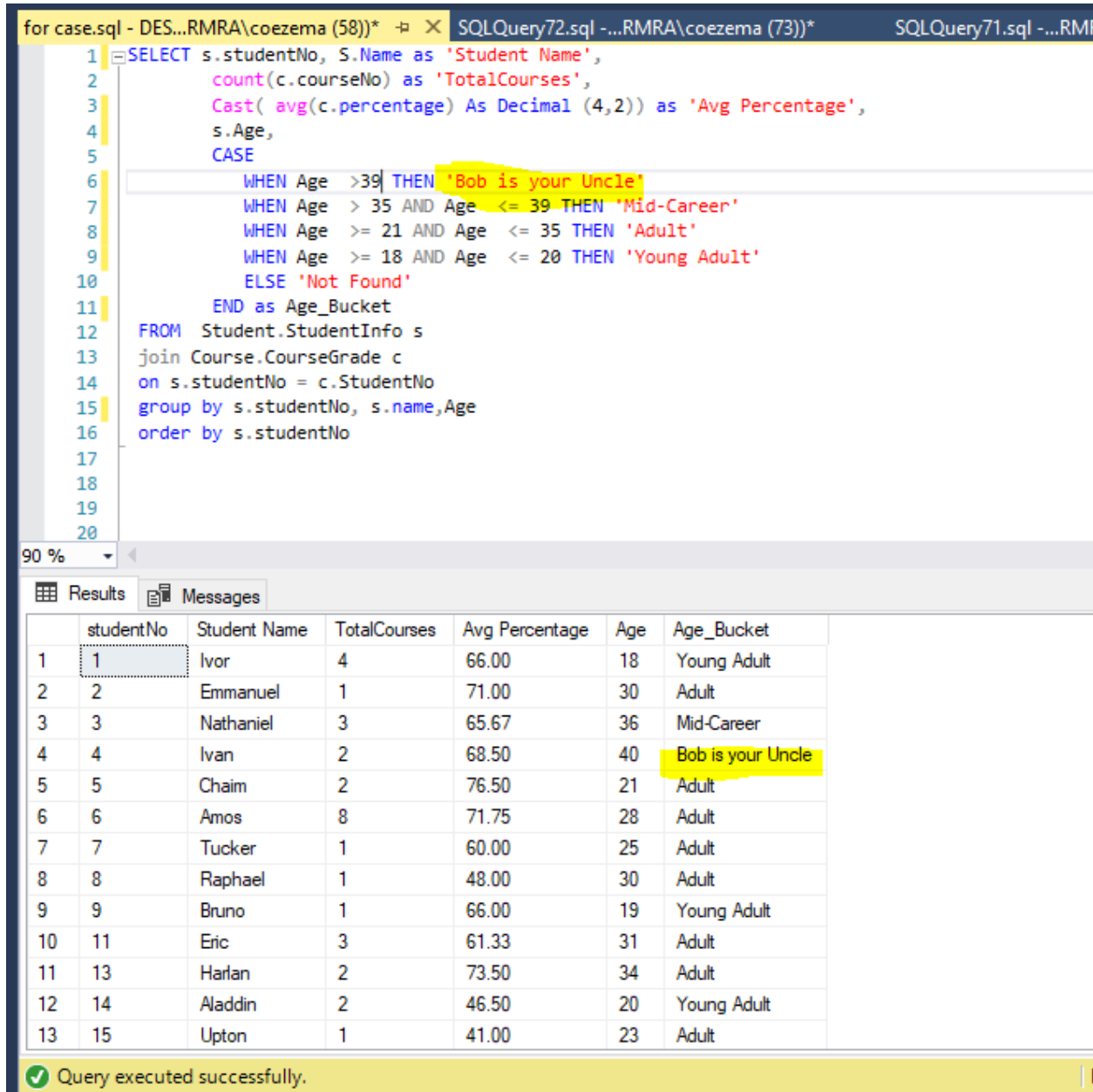
#### 4. Create any query using a set of Case Expressions

##### User Story:

As the college admission committee, I want to know which age\_bucket a student belongs so that I can develop dynamic reports with that data set.

User story requirements:

- Insert age bucket column for report developers to use



```
for case.sql - DES...RMRA\coezema (58))* X SQLQuery72.sql -...RMRA\coezema (73))* SQLQuery71.sql -...RMRA\coezema (73))*
1 SELECT s.studentNo, S.Name as 'Student Name',
2     count(c.courseNo) as 'TotalCourses',
3     Cast( avg(c.percentage) As Decimal (4,2)) as 'Avg Percentage',
4     s.Age,
5     CASE
6         WHEN Age >39 THEN 'Bob is your Uncle'
7         WHEN Age > 35 AND Age <= 39 THEN 'Mid-Career'
8         WHEN Age >= 21 AND Age <= 35 THEN 'Adult'
9         WHEN Age >= 18 AND Age <= 20 THEN 'Young Adult'
10        ELSE 'Not Found'
11    END as Age_Bucket
12 FROM Student.StudentInfo s
13 join Course.CourseGrade c
14 on s.studentNo = c.StudentNo
15 group by s.studentNo, s.name, Age
16 order by s.studentNo
17
18
19
20
```

	studentNo	Student Name	TotalCourses	Avg Percentage	Age	Age_Bucket
1	1	Ivor	4	66.00	18	Young Adult
2	2	Emmanuel	1	71.00	30	Adult
3	3	Nathaniel	3	65.67	36	Mid-Career
4	4	Ivan	2	68.50	40	Bob is your Uncle
5	5	Chaim	2	76.50	21	Adult
6	6	Amos	8	71.75	28	Adult
7	7	Tucker	1	60.00	25	Adult
8	8	Raphael	1	48.00	30	Adult
9	9	Bruno	1	66.00	19	Young Adult
10	11	Eric	3	61.33	31	Adult
11	13	Harlan	2	73.50	34	Adult
12	14	Aladdin	2	46.50	20	Young Adult
13	15	Upton	1	41.00	23	Adult

Query executed successfully.

Results:

- We were able to add the age column for report developers.